MAX: A Multi Objective Memory Architecture eXploration Framework for Embedded Systems-on-Chip

T.S. Rajesh Kumar **C.P. Ravikumar**
Texas Instruments India Ltd.
C.V. Raman Nagar
Bangalore, 560 091, India

{tsrk,ravikumar}@ti.com

R. Govindarajan
Supercomputer Education & Research Centre
Indian Institute of Science
Bangalore, 560 012, India

govind@serc.iisc.ernet.in

Abstract— Today's feature-rich multimedia products require embedded system solution with complex System-on-Chip (SoC) to meet market expectations of high performance at a low cost and lower energy consumption. The memory architecture of the embedded system strongly influences these parameters. Hence the embedded system designer performs a complete memory architecture exploration. This problem is a multi-objective optimization problem and can be tackled as a two-level optimization problem. The outer level explores various memory architecture while the inner level explores placement of data sections (data layout problem) to minimize memory stalls. Further, the designer would be interested in multiple optimal design points to address various market segments. However, tight time-to-market constraints enforces short design cycle time. In this paper we address the multi-level multi-objective memory architecture exploration problem through a combination of Multi-objective Genetic Algorithm (Memory Architecture exploration) and an efficient heuristic data placement algorithm. At the outer level the memory architecture exploration is done by picking memory modules directly from a ASIC memory Library. This helps in performing the memory architecture exploration in a integrated framework, where the memory allocation, memory exploration and data layout works in a tightly coupled way to yield optimal design points with respect to area, power and performance. We experimented our approach for 3 embedded applications and our approach explores several thousand memory architecture for each application, yielding a few hundred optimal design points in a few hours of computation time on a standard desktop.

I. Introduction

Today's VLSI technology allows to integrate tens of processor cores on the same chip along with embedded memories, application specific circuits, and interconnect switches. Such devices are being used in feature-rich multimedia applications such as mobile cameras. The key to the success of such systems-on-chip (SoC) which are targeted for commodity market will be to achieve low cost, high performance, and low power dissipation.

One of the key factors that drives the cost and power dissipation of an embedded system-on-chip is the memory architecture.

The memory architecture of embedded DSPs are complex and custom designed to improve the run-time performance and power consumption. The on-chip memory can be SRAM, and/or ROM and/or embedded DRAM and similarly the off-chip memory can be DRAM and/or SRAM. The on-chip memory referred as Scratch Pad memory is organized into multiple memory banks to facilitate multiple simultaneous data accesses. Further on-chip memory bank can be a single-access RAM (SARAM) or a dual-access RAM (DARAM), to provide single or dual access to the same memory bank in a single cycle. Also the on-chip memory banks can be of different sizes. Smaller memory banks consume lesser power per access than the larger memories. This architecture presents opportunities to optimize power consumption by placing the most frequently accessed data variables in the smallest bank size.

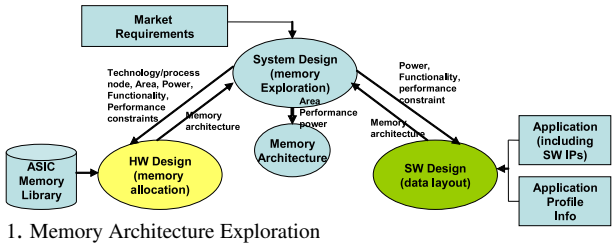

Fig. 1. Memory Architecture Exploration

Figure 1 presents the traditional memory exploration process. At the memory exploration step, a memory architecture must be defined which includes determining the on-chip memory size, the number and size of each memory bank, the number of memory ports per bank, the types of memory (scratch pad RAM or cache), wait-states/latency, we refer to this as logical memory architecture as it does not tie the architecture at this point to a specific ASIC memory library module. Each of the memory architecture is evaluated based on the performance, VLSI area and power consumption.

To get the memory power and area numbers for a given memory architecture, the logical memories have to be mapped to physical memory modules available in an ASIC memory

library for a specific technology/process node. A number of approaches have been proposed for mapping logical memory to physical memories [13, 12]. Each of the logical memory bank can be implemented physically in many ways. For example, for a logical memory bank of 4K*16 bits can be formed with two physical memories of size 2K*16 bits or four physical memories of size 2K*8 bits. The memory allocation problem in general is NP-Complete [13].

To evaluate a memory architecture with respect to performance and minimize the number of memory stalls, the data buffers of the application need to be placed carefully in different types of memory; this is known as the **data layout problem**. Typically the data layout is performed manually and hence takes a significant amount of time (approximately 1-2 man months). For a given memory architecture and application, the data layout optimizes the run-time performance and power consumption by careful placement of application's data buffers in different memory banks.

While defining a memory architecture the system architects work with the hardware and software design teams independently. The hardware design teams are consulted for the area and power targets and the software design team is consulted on the performance and power. This iteration between the hardware and software design while balancing the system requirements, as shown in Figure 1, consumes lot of time and because of the tight time to market constraints the system design process is cut short resulting in sub-optimal memory architecture and over design. There is a fine trade-off involved and there are many design points (memory architectures) which are Pareto optimal with respect to Memory performance (stall cycles), Memory Power and Memory area. The system designers need to explore all the Pareto optimal points and based on the market requirements choose a design point that satisfies the overall system requirements. To be able to do this, the memory architecture exploration has to be done along with memory allocation and data layout. In this paper we propose an integrated memory architecture exploration framework that performs the memory architecture exploration along with memory allocation and data layout.

A. Related Work

The data layout problem [10, 7, 6, 9, 1, 2] has been widely researched in the literature both from a performance and power perspective individually. The authors of [7, 6] have formulated data layout problem as Integer Linear Programming (ILP), memory architecture is on-chip memory and off-chip memory with different latencies. [9, 1] have addressed the partitioning of simultaneously accessed data variables in multiple single-port memory banks to avoid memory bank conflicts. [11] handles the data partitioning problem for self-conflicting data variables (variable that are accessed multiple times in the same cycle and will result in memory conflict if placed in single-port memory). All these work has addressed the data layout problem for a given memory architecture with performance as the objective.

Memory Architecture Exploration is performed in [3] using a low-energy memory design method, referred as VAbM, that optimizes the memory area by allocating multiple memory banks with variable bit-width to optimally fit the application data. In [2], Benini et al., present a method that combines the memory allocation and data layout together to optimize the power consumption and area of memory architecture. They start from a given data layout and design smaller (larger) bank size for the most (least) frequently accessed memory addresses. In our method the data layout is not fixed and hence it explores the complete design space with respect to performance and power. Performance-energy design space exploration is presented in [5]. They present a branch and bound algorithm that produces Pareto trade-off points representing different performance-energy execution options. [4] presents an integrated memory exploration approach that combines scheduling and memory allocation. They consider different speed of memory accesses during memory exploration. They consider only performance and area as objectives and they output only one design point. In our work we consider area, power and performance as objectives and we explore the complete design space to output several hundreds of Pareto optimal design points.

[15] and [16] addresses the memory hierarchy exploration problem in the Genetic Algorithm framework; their target architecture consists of separate L1 caches for Instruction and data, unified L2 cache and bus sizes for L1-to-L2 and L2-to-external memory. Their objective function is a single formula which combines area, average access time and power. In [15], additional parameters such as bus width and bus encoding are considered, and the problem is modeled in multi-objective GA framework. Our work addresses the memory architecture exploration of DSP memory architecture that is typically organized as multiple memory banks where each of the banks can consist of single/dual port memories with different sizes. We consider non-uniform memory bank sizes. Our work uses an integrated data-layout and memory architecture exploration approach, which is key for guiding GA's search path in the right direction. The cost functions and the solution search space will be very different for a cache based memory architecture used in [15, 16] and an on-chip scratch pad based DSP memory architecture used in our paper.

In this paper we formulate the memory architecture exploration with data layout as an integrated multi objective formulation. For memory architecture exploration with physical memory mapping, we use a multi-objective non-dominated sorting Genetic Algorithm approach [14]. For the data layout problem which needs to be solved for each of the 1000s of memory architecture, we propose a fast efficient heuristic method that obtains near optimal data layout. Thus the overall framework uses a two level iterative approach with memory architecture exploration at the outer level and data layout at the inner level.

The main contributions of this paper are (a) presenting a framework (MAX) that integrates the memory architecture exploration, memory allocation and data layout procedures to-

gether to explore the memory design space for a DSP processor based Systems On a Chip (b) model the memory architecture exploration as a multi objective Genetic Algorithm (GA) with performance, power and area as the objectives. Our method optimizes the memory architecture, for a given application, and presents a set of solution points that are optimal with respect to performance, power and area. (c) proposing an efficient heuristic that addresses the data layout problem for memory architecture with multiple memory banks, dual-ported memory, different latencies and on-chip/off-chip partitioning.

The remainder of this paper is organized as follows. In Section 2, we present the problem definition. In Section 3, we present our MAX framework and the data layout heuristic algorithm. In Section 4, we present the experimental methodology and results. Finally in Section 5, we conclude and outline some of the future work.

II. Multi Objective Memory Architecture eXploration (MAX) Framework

A. Method Overview

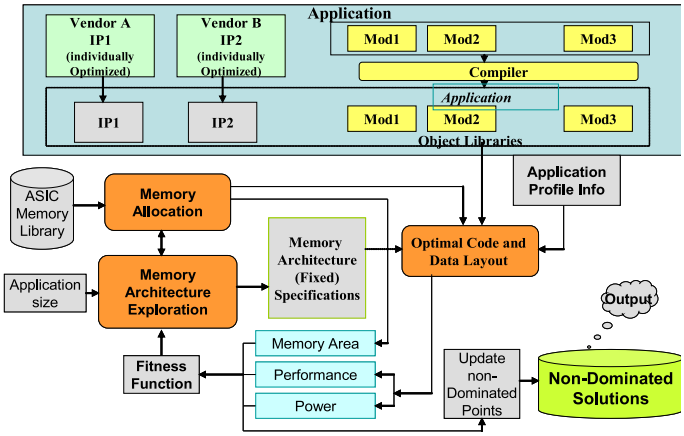

Fig. 2. MAX: Memory Architecture eXploration Framework

We formulate the memory architecture exploration problem as a multi-objective GA [8] to obtain the set of *Pareto optimal* design points. The multiple objectives are minimizing memory stall cycles, memory area and memory power. Figure 2 explains our MAX framework. The core engine of the framework is the multi-objective memory architecture exploration, which is implemented as a non-dominated sorting Genetic Algorithm (GA) [14]. The memory exploration takes the application data size as the input and forms different memory architectures by picking the memory modules from the ASIC memory library. The memory mapping procedure builds the complete memory architecture from the memory modules chosen by the exploration block. If the memory modules together does not form a proper memory architecture, the memory mapping block rejects the chosen memory architecture as invalid. Also the memory mapping block checks the access time of the on-chip memory modules against the processor cycle time. The ASIC memory library contains the active power, leakage power, access time and memory area for all the memory modules. Once the memory modules are selected the memory mapping block computes the total memory area, which is the sum of all the individual memory modules.

Details on the selected memory architecture like the on-chip memory size, number of memory banks, number of ports, off-chip memory bank latency are passed to the data layout procedure. The application data buffers and the application profile information also given as inputs to the data layout. The application itself consists of multiple modules, including several third-party IP modules as shown in Figure 2. With these inputs the data layout maps the application data buffers to the memory architecture; the data layout heuristic is explained in Section IV. The output of data layout is a valid placement of application data buffers, memory performance (memory stalls) and the memory power. The fitness function for the memory exploration is computed With the memory area, performance and power.

Based on the fitness function the GA evolves by selecting the fittest individuals (near optimal memory architectures) to the next generation. Since the fitness function contains multiple objectives, the fitness function is computed by ranking the chromosomes based on the non-dominated criteria (detailed explanation provided in Section III). This process is repeated for a maximum number of generations specified as a input parameter.

III. Genetic Algorithm Formulation

To map an optimization problem to the GA framework, we need the following: chromosomal representation, fitness computation, selection function, genetic operators, the creation of the initial population and the termination criteria.

A. Chromosome Representation

For the memory architecture exploration problem, each individual chromosome represents a physical memory architecture. A chromosome consists of two parts (a) number of logical memory banks (N_l), and (b) list of physical memory modules that form the logical memory bank. We have limited the number of physical memory modules per logical memory bank to at most k. As an example, if the logical bank is of size 8K*16bits then, the physical memory modules can be two 4K*16bits or four 2K*8bits and so on. Thus, a chromosome is a vector of d elements, where $d = N_l * k + 1$ and N_l is the number of logical memory banks. The first element of a chromosome is N_l and it can take value in (0 .. $maxbanks$), where $maxbanks$ is the maximum number of logical banks given as input parameter. The remaining elements of a chromosome can take a value in (0 .. m), where $1..m$ represent the physical memory module id in the ASIC memory library. Note that the length of the chromosomes differ based on the N_l.

B. Decoding a Chromosome

Each chromosome is a pair of $< N_l, k * N_l >$. For decoding a chromosome, first N_l is read and then for each of the N_l logical banks, the chromosome has k elements. Each of the k elements are integers used to index into ASIC memory library. With the k physical memory modules, a rectangular logical memory bank is formed. We have used a memory allocator that performs exhaustive combinations with the k physical memory modules to get the largest logical memory with the required word size which is specified as an input parameter. In this process it may happen that some of the physical memory modules may be wasted. For example, if k=4, then if the 4 elements are 2K*8bits, 2K*8bits, 1K*8bits, and 16K*8bits and if the bit-width requirement is 16-bits then our memory allocator builds a 5K*16bits logical memory bank from the given 4 memory modules. Note that 11K*8bits is wasted in this configuration and this architecture will have a low fitness as the memory area will be very high. The memory area of a logical memory bank is the sum of the memory area of all the physical memory modules. This process is repeated for each of N_l logical memory banks. The memory area of a memory architecture is the sum of the area of all the logical memory banks.

C. Chromosome Selection and Generation

The strongest individuals in a population are used to produce new off-springs. The selection of an individual depends on its fitness, an individual with a higher fitness has a higher probability of contributing one or more offspring to the next generation. In every generation, from the P individuals of the current generation, M more offspring are generated using mutation and crossover operators, resulting in a total population of $(P + M)$. From this P fittest individuals survive to the next generation. The remaining M individuals are annihilated. Crossover and mutation operators are implemented in standard way.

D. Fitness Function and Ranking

For each of the individuals, the fitness function computes M_{area} M_{pow} and M_{cyc}. The value of M_{cyc} is computed by data layout using the heuristic explained in section IV. The M_{area} is obtained from the memory mapping block, which is the sum of area of all the memory modules used in the chromosome.

Memory Power corresponding to a chromosome is computed as follows. The data layout heuristic places all the data buffers to memory banks. The data placement information is represented by the variable I_{ij}, where $I_{ij} = 1$, if data variable i is placed in memory bank j else $I_{ij} = 0$. If p_j is the power per access of memory bank j, and AF_i is the number of times data variable i is accessed then the total power P_j for memory bank j is

$$P_j = \sum_{i=1}^{d} p_j * AF_i * I_{ij}$$

Thus the total power P_t for all the memory banks is the sum of all the individual memory bank's power.

$$M_{pow} = \sum_{j=1}^{N_b} P_j$$

where N_b is the total number of banks including off-chip memory.

Once the memory area, memory power and memory cycles are computed for all the individuals in the population, individuals are ranked according to the *Pareto optimality* conditions given in the following Equation. Let $(M^a_{pow}, M^a_{cyc}, M^a_{area})$ and $(M^b_{pow}, M^b_{cyc}, M^b_{area})$ be the memory power, memory cycles and memory area of chromosome A and chromosome B, A dominates B if the following expression is true.

$$\begin{aligned}&(((M^a_{pow} < M^b_{pow}) \wedge (M^a_{cyc} \leq M^b_{cyc}) \wedge (M^a_{area} \leq M^b_{area}))\\ &\vee((M^a_{cyc} < M^b_{cyc}) \wedge (M^a_{pow} \leq M^b_{pow}) \wedge (M^a_{area} \leq M^b_{area}))\\ &\vee((M^a_{area} < M^b_{area}) \wedge (M^a_{cyc} \leq M^b_{cyc}) \wedge (M^a_{pow} \leq M^b_{pow})))\end{aligned}$$

The ranking process in multi-objectctive GA proceeds as follows. All non-dominated individuals in the current population are identified and flagged. These are the best individuals and assigned a rank of 1. These points are then removed from the population and the next set of non-dominated individuals are identified and ranked 2. This process continues until the entire population is ranked. Fitness values are assigned based on the ranks. Higher fitness values are assigned for rank-1 individuals as compared to rank-2 and so on. This fitness is used for the selection probability. The individuals with higher fitness gets a better chance of getting selected for reproduction.

One of the common problems in multi-objective optimization is solution diversity [8]. Basically the search path may progress towards only certain objectives resulting in design points favoring those objectives, causing a skew in the design points explored. In order to get a good distribution of solutions in the Pareto-optimal front, the fitness value is reduced for solution that has many neighboring solutions. We use the sharing function method explained in [14]. During the ranking process, all individuals with the same rank are assigned equal fitness. To maintain solution diversity, the fitness values for solutions are adjusted based on the number and proximity of neighboring solutions (this is known as the *niche count* [8]). More the neighboring solutions, lesser the fitness value.

The GA must be provided with an initial population that is created randomly. In our implementation we have used a fixed number of generations as the termination criterion.

IV. Heuristic Algorithm

In this section we describe a 3-step heuristic method for data placement. As mentioned earlier the data layout problem is NP Complete. Further we want to solve the data placement for all the memory architectures considered in the exploration

stage, which could be in several thousands. Using exact solving method such as Integer Linear Programming (ILP) for this problem may be prohibitively expensive. Even an evolutionary approach, such as GA or SA can take as much as 5 to 10 minutes of computation time for each data layout problem.

A. Data Partitioning into internal and external memory

The first step in data layout is to identify and place all the critical data sections in the internal memory. Data sections are sorted in descending order of frequency per byte (FPB_i). Based on the sorted order, data sections are identified for placement in internal memory till free space is available. We refer all the on-chip memory banks together as internal memory. Note that the data sections are *not* placed at this point but only identified for internal memory placement. The actual placement decision are taken later as explained below.

Once all the data sections to be placed in internal memory are identified, the remaining sections are placed in external memory. The cost of placing data section i in external memory is computed by multiplying the access frequency of data i with the wait-states of external memory. The placement cost is computed for all the data sections placed in the external memory.

B. DARAM and SARAM placements

The objective of the next two steps is to resolve as many conflicts (self-conflicts and parallel-conflicts) by utilizing the DARAM memory and the multiple banks of SARAM. Self-conflicts can only be avoided if the corresponding data section is placed in DARAM. On the other hand, parallel conflicts can be avoided in two ways either by placing the conflicting data a and b in two different SARAM banks or by placing the conflicting data a and b in any DARAM bank. But former solution is attractive as the SARAM area cost is much less compared to DARAM area cost. Considering that self-conflicts can only be avoided by placing in DARAM and the cost of DARAM is very high, data placement decisions in DARAM needs to be done very carefully. Also, many of the DSP applications have large self-conflicting data and the DARAM placements is very crucial for reducing run-time of the application.

The heuristic algorithm considers placement of data in DARAM as the first step within internal memory placements. Data sections that are identified for placement in internal memory are sorted based on the self-conflict per byte (SPB_i). Data sections in the sorted order of SPB_i are placed in DARAM memory until all DARAM banks are exhausted. Cost of placing data section i in DARAM is computed and added as part of the overall placement cost.

Once the DARAM data placements are complete, SARAM placement decisions are made. Parallel conflicts between data sections i and j can be resolved by placing conflicting data sections in different SARAM banks. The SARAM placement is started by sorting all the data sections identified for placement in internal memory based on the total number of conflicts (TC_i). Note that all data sections including the ones that are already placed in DARAM are considered while sorting for SARAM placement. This is because the data placement in DARAM is only tentative and may be backtracked in the backtracking step if there is a larger gain (i.e., more parallel conflicts resolved) in placing a data section i in DARAM instead of one or more data sections already placed in DARAM. Initially if a data section is already placed in DARAM then it is ignored and the next data section in the sorted order is considered for SARAM placement. The placement cost for placing data section i in SARAM bank b computed considering all the data sections already placed in DARAM and SARAM banks. Among this the memory bank b that results in the minimum cost is chosen.

Next, the heuristic backtracks to find if there is any gain in placing data i in DARAM by removing some of the already placed data sections from DARAM. This is done by considering the data size of i and the minimum placement cost of data i in SARAM. If there is one of more data sections (refer this set of data as *daram-remove-set*) in DARAM with size more than the size of data section i **and** the sum of self-conflicts for all these data sections are less than the minimum placement cost of data i in SARAM, then there can potentially a possibility of gain of placing data i into DARAM by removing the *daram-remove-set*. Note that it is only a possibility and not a certain gain.

Once a *daram-remove-set* is identified, to ensure that there is gain in the backtracking step, the data sections that are part of *daram-remove-set* needs to be placed in SARAM banks and minimum placement cost has to be computed again for each of the data section. If the sum of the minimum placement cost for all data sections in *daram-remove-set* is greater than the original minimum cost of data i, then there is no gain in backtracking and data section i is placed in SARAM banks. Else there is gain in backtracking and the *daram-remove-set* is removed from DARAM and placed in SARAM. Data section i is placed in DARAM. The overall-placement cost is updated. This process is repeated for all data sections identified to be placed in the internal memory. The overall-placement cost gives the memory cycles (M_{cyc}) for placing application data for a given memory architecture.

V. Experimental Results

A. Experimental Methodology

We have used Texas Instrument's TMS320C55X and Texas Instrument's Code Composer Studio (CCS) environment for obtaining the profile data and also for validating the data-layout placements. We have used 3 different applications from the multimedia and communications domain as benchmarks The kernels of the applications are developed in hand-optimized assembly code. The applications are compiled with the C55X processor compiler and assembler. The profile data is obtained by running the compiled executable in a cycle accurate software simulator. For obtaining conflict data we used one large bank of single-access RAM that fits the application data size. This configuration is selected because this does not

resolve any of the parallel or self conflicts. The output profile data contain (a) frequency of access for all data sections (b) the conflict matrix. The other inputs required for our method is the application data section sizes, which are obtained from the C55X linker. We have used TI's ASIC memory library for the memory allocation step. The area and power numbers are obtained from the ASIC memory library. [1]

B. Experimental Results

This section presents the experimental results on the multi-objective Memory Architecture Exploration. The objective is to explore the memory design space to obtain all the non-dominated design solutions (memory architectures) that are Pareto optimal with respect to area, power and performance.

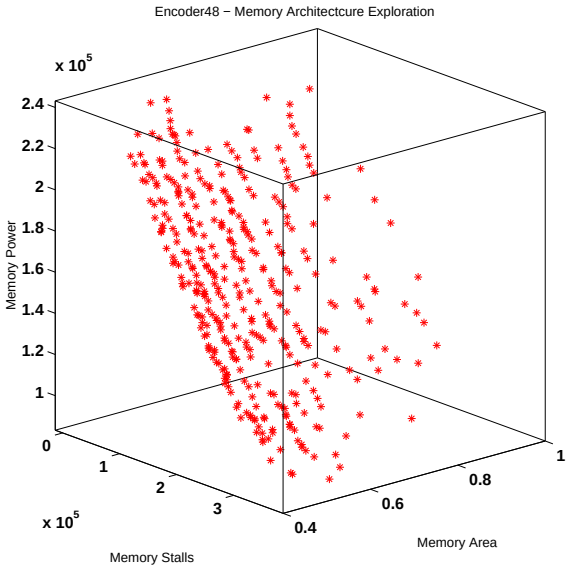

Fig. 3. Voice Encoder (3D view): Memory Architecture Exploration

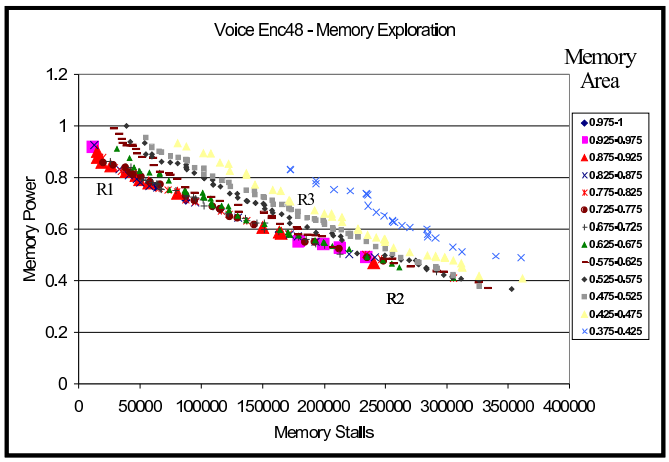

Fig. 4. Voice Encoder: Memory Architecture Exploration

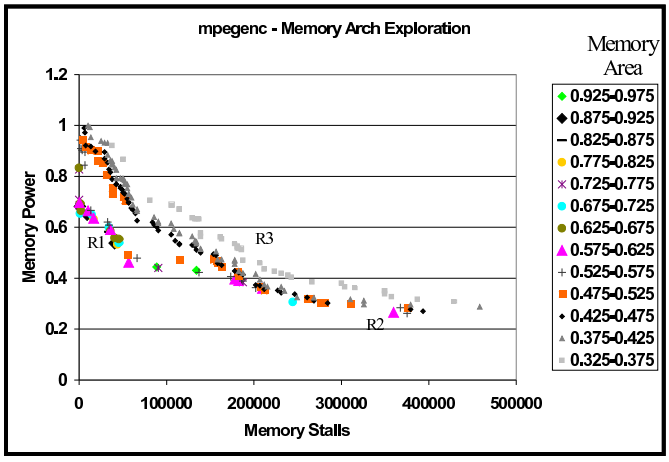

Fig. 5. MPEG Encoder: Memory Architecture Exploration

The Pareto-optimal design points identified by our framework for the voice encoder application is shown in Figure 3. [2]. It should be noted that the non-dominated points seen by the multi-objective GA are only near optimal, as the evolutionary method may result in another design point in future generations that could dominate. One can observe a set of points for each x-z plane (memory power - memory stalls) corresponding to a given area. These represent the trade-off in power and performance for a given area. The same graph is plotted in a 2D-graph in Figure 4 where architecture which require an area within a specific range are plotted using the same color. These correspond to the points in a set of x-z planes for the area range.

Figures 5, 4 and 6 show the set of non-dominated points each correspond to a *Pareto Optimal* Memory Architecture for the 3 applications. It can be observed from Figures 5, 4 and 6 that the increase in memory area results in improved performance and power. Increased area will translate to more on-chip memory, increased number of memory banks, and more dual-port memory — all these are essential for improved performance. We look at the optimal memory architectures derived by our framework. In particular we consider (a) R1, (b) R2 and (c) R3 in each of the figures. The region R1 corresponds to (high performance, high area, high power); R2 corresponds to (low performance, high area, low power); and the region R3 corresponds to (medium performance, low area, medium power). Since the memory exploration design space is very large, it is important to focus on regions that is critical to the targeted applications. The region R1 has memory architectures with large dual-port memory that aids in improved performance but also a cause for high power consumption. The region R2 has large number of memory banks of different size. This helps in reducing the power consumption by keeping the data sections with

[1] As the ASIC library is proprietary to Texas Instruments, we present only the normalized power and area numbers

[2] We recommend this page be viewed in color for better readability of the graphs

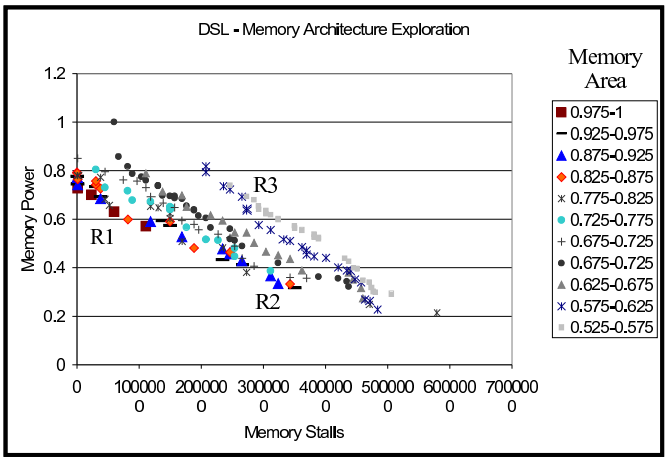

Fig. 6. DSL: Memory Architecture Exploration

Application	Time Taken	No of Arch explored	No of non-dominated points
Mpeg Enc	2.5 hours	9780	670
Vocoder	3.5 hours	13724	981
DSL	2 hours	7240	438

TABLE I
MEMORY ARCHITECTURE EXPLORATION

higher access frequency to smaller memory banks. But the region R2 does not have dual-port memory modules and hence results in low performance. But the presence of higher number of memory banks increases the area. The region R3 does not have dual port memory modules and also has lesser number of on-chip memory banks. Since the memory banks are large, the power per access is resulting in higher power consumption. Note that for a given area there can be more than one memory architectures. Also it can be observed that for a fixed memory area, the design points are pareto optimal with respect to power and performance. Observe the wide range of trade-off available between power and performance for a given area. We observe that By trading off performance power consumed can be reduced as much as 70-80%.

Table I gives details on the run-time, the total number of memory architectures explored and the number of non-dominated (near-optimal) points for each of the application. Note that even the number of non-dominated design solutions is also large. Hence to select a optimal memory architecture for a targeted application, the system designer needs to follow a clear top down approach of narrowing down the region (area, power, performance) of interest and then focus on specific memory architectures.

VI. CONCLUSION AND FUTURE WORK

In this paper we have presented our Memory Architecture eXploration (MAX) framework that integrates memory exploration, logical to physical memory mapping and data layout. We have addressed three of the key system design objectives (i) memory area, (ii) performance and (iii) memory power. Our approach explores the design space and gives a few hundred pareto-optimal memory architectures at various system design points in a few hours of run time. We have presented a fully automated approach that meets the time-to-market requirements. As a future work we intend to extend our MAX framework by considering multi-banked SDRAM memory and caches.

REFERENCES

[1] M.Ko, S.S.Bhattacharyya. Data Partitioning for DSP Software Synthesis. In Proceedings of the International Workshop on Software and Compilers for Embedded Processors, Sep-2003.

[2] L.Benini, L.Macchiarulo, A.Macii, M.Poncino. Layout Driven Memory Sunthesys for Embededed Systems-on-Chip. CASES 2002.

[3] Y.Cao, H.Tomiyama, T.Okuma, H.Yasuura, "Data Memory Design Considering Effective Bitwidth for Low Energy Embedded Systems," *ISSS 2002*.

[4] J.Seo, T.Kim, and P.Panda. An Integrated algorithm for memory allocation and assignment in high-level synthesis. In *Proc. of 39th Design Automation Conf., pages 608-611, 2002.*

[5] R.Szymanek, F.Catthoor, and K.Kuchcinski. Time-Energy Design Space Exploration for Multi-Layer Memory Architectures. *DATE 2004*.

[6] J.Sjodin, and C.Platen. Storage Allocation for Embedded Processors. CASES 2001.

[7] O. Avissar, R. Barua, D. Stewart. Heterogeneous Memory Management For Embedded Systems. CASES 2001, Nov 2001.

[8] D.E. Goldberg. Genetic Algorithms in Search, Optimizations and Machine Learning. Addison-Wesley, 1989.

[9] R. Leupers, D. Kotte. Variable Partitioning for Dual Memory Bank DSPs. International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Salt Lake City (USA), May 2001.

[10] P.R. Panda, N.D.Dutt, and A.Nicolau. On-chip vs. off-chip memory: The data partitioning problem in embedded processor-based systems. ACM Trans. Design Automation of Electronic Systems, 5(3):682–704, July 2000.

[11] M.A.R. Saghir, P.Chow, C.G.Lee. Exploiting Dual Data-Memory Banks in Digital Signal Processors. In Proc. of the 7th Intl Conf. ASPLOS, pp.234-243, October 1996.

[12] H.Schmit and D.Thomas, "Array Mapping in Behavioral Synthesis," *ISSS 95*.

[13] Pradip K.Jha and Nikil D.Dutt, "Library Mapping for Memories,", *EuroDesign 1997*.

[14] Deb.K., Goel T. Controlled elitest non-dominated sorting genetic algorithms for better convergence. In Porceedings of Evolutionary Multi-Criterion Optimization. 67-81, 2001.

[15] M.Palesi, T.Givargis. Multi-objective Design Space Exploration Using Genetic Algorithms. International Workshop on Hardware/Software Codesign (CODES), May 2002.

[16] Ascia, G., Catania, V., and Palesi, "Parameterised system design based on genetic algorithms", CASES 2001.